Living Atlas

Final Report

Center for Environmental Research, Education, and Outreach (CEREO)

WASHINGTON STATE UNIVERSITY

Living Atlas Development Team

Long, Joshua

Kolb, Mitchell, William

Svetlik, Sierra Amelia

Fall 2023

I. Introduction

This document serves as an update of CEREO Living Atlas's progress on the Living Atlas app. It will cover the current progress of CLA on the Living Atlas app. Included in this document will be details on the project and team members, Living Atlas specifications and requirements, our approach to solving the given the issue, and the plans for testing the app. This document will also provide updates of the project and details of the progress made.

I.1. Project Introduction

CLA has been tasked with making an app that will be known as Living Atlas. The Living Atlas app will be able to collect environmental information about the Columbia River Basin of many different types, as well as having this information accessible by anyone. The ability to input information in the app will be restricted, but the ability to view the information will not be. The target groups for the app includes researchers, locals, tribes, and government officials. The end goal of this project is to have a repository of a wide variety of environmental data that can be accessed by anyone and uploaded and edited by anyone with permission.

I.2. Background and Related Work

The Center for Environmental Research, Education, and Outreach (CEREO) is interested in collecting information about the Columbia River Basin, but there is no good way to record and access the many different types of information that they are interested in. They want a wide variety of individuals, companies, and organizations to be able to input, edit, and access the information on the app.

The app can be compared to Zillow, but for environmental information. The user should be able to pull up a map and be able to see the information, and sort by types and categories of information.

Other websites that the app can be compared to include StoryMaps and a government data repository. StoryMaps has a map feature, and the data repository has a search feature. Both would be desirable in our app.

I.3. Project Overview

Living Atlas is a web application aimed at solving the problem of scattered and inaccessible environmental data. Our goal is to provide a central location for collecting, sharing, and accessing environmental data, making it available to a wide range of stakeholders including tribal communities, academic institutions, and government agencies.

The platform will allow for easy viewing by everyone but will require authentication for uploading data to ensure the validity and accuracy of the information being shared. Additionally, the platform will be able to connect to external sources, further expanding the range of environmental data available on the platform.

Living Atlas will offer the following key features and functionality:

1. Data Collection: A user-friendly interface for collecting and uploading environmental data from various sources, including individuals, institutions, and government agencies. The uploader will first be verified to ensure the data's accuracy and validity.

2. Data Management: An efficient and organized system for managing and storing the collected data, making it easily accessible and searchable. Users will be able to sort, filter, and analyze the data to gain valuable insights.

3. Data Sharing: A platform for sharing environmental data with the wider community, promoting collaboration, and improving access to data. The platform will be designed to be easily accessible and viewable by anyone.

4. External Data Connection: The ability to connect to external data sources, expanding the range of environmental data available on the platform.

5. Data Visualization: An interactive map-based interface for visualizing environmental data. The map-based interface will allow users to view data in a geographical context, providing valuable insights and promoting a deeper understanding of the environment.

Team Structure:

The Living Atlas project will be divided into two teams, each responsible for a specific aspect of the project. The clear division of roles and responsibilities between the Platform Team and Visualization Team is a key aspect of the project's success. This team structure helps to ensure the efficient and effective delivery of the Living Atlas project. Each team can focus on its specific tasks and collaborate effectively to achieve the overall goals of the project.

Living Atlas1: Platform Team

- The Platform Team will handle the data collection, management, and sharing aspects of the Living Atlas project. This includes creating a user-friendly interface for collecting data, an efficient system for managing data, a platform for sharing data, and the ability to connect to external data sources. The goal is to provide a comprehensive solution for collecting, managing, and sharing environmental data.

Living Atlas2: Visualization Team

- The Visualization Team will be responsible for creating the data visualization component of the Living Atlas project. They will design and develop interactive and user-friendly visualizations of the environmental data collected by the platform. The Visualization Team will work closely with the Platform Team to ensure that the visualizations are aligned with the platform's data management and sharing features. The ultimate goal is to make the environmental data easily accessible and understandable for the platform's users.

In conclusion, Living Atlas aims to make a significant impact by providing a central hub for environmental data, enabling better decision-making and promoting collaboration among a diverse group of stakeholders. We are committed to making this project a success and are excited to see the positive impact it will have on the environmental community.

I.4. Client and Stakeholder Identification and Preferences

Our Client is the Center for Environmental Research, Education, and Outreach (CEREO) team. CEREO seeks to apply innovative technologies and management tools to the ever-growing challenges of climate change and environmental sustainability. The CEREO team has a network of 350 faculty, staff, students, and outside industry leaders. They operate as a clearinghouse for a wide range of projects such as watershed management, tracking the nitrogen cycle, and studying urban socio-ecological systems. The subteam that we will be working with is the co-directors of CEREO, Dr. Jan Boll and Dr. Julie Padowski, and some additional core faculty/staff such as Dr. Hannah Haemmerli and principal assistant Jacqueline Richey McCabe.

The users of the Living Atlas website are the CEREO team and the National Science Foundation Traineeship Program which has partnered with CEREO. Other users include tribal communities, academic institutions, and government agencies. The client has stated that they intend to use this website to help share documents so the potential users are the future affiliates of the CEREO team. Possible other parties that relate to this project could be maintenance/servicers that host the website once it is operational and deployed. Cloud-based workers could also be included if when the website is deployed it is through a cloud service provider.

To allow for efficient communication, the CEREO team requests a web application that uses an interactive map and database to store and display information regarding rivers, watersheds, and communities in the larger pacific northwest region focusing around the Columbia river basin. This application will require the implementation of a well-structured and organized user interface and filtering system. Living Atlas team 1 will focus on the database and backend of the web application while Living Atlas team 2 will focus on the user interface and interactive map. This web application is intended to solve the problem that there is no one place to collect and share environmental data.

Lastly, the CEREO team requests that the web application be accessible to all users who wish to use it. Although it will be viewable to all, users who wish to contribute to the data shown on this web application will have to be approved by the CEREO team. The Living Atlas teams must build a web application with these preferences in mind.

II. Team Members - Bios and Project Roles

Joshua Long is a computer science major with a passion for software development. He is pursuing a Bachelor of Science in Computer Science and is eager to apply his technical skills to real-world problems. Joshua has experience with programming languages such as C#, JavaScript, and Python. He is dedicated to becoming a software developer and is eager to make a positive impact on the industry. Joshua is also the team lead of Living Atlas 1 which is responsible for handling the data collection, management, and sharing aspects of the Living Atlas project.

Mitchell Kolb is a senior majoring in computer science interested in machine learning and mobile/desktop application development. The languages Mitchell is most skilled in include C/C++, Python, HTML/CSS, and SQL. He has knowledge of technologies/frameworks such as

Git, Flask, Angular, PyTorch, and Pandas. Mitchell's responsibilities include assisting his team with the creation of a database and backend for the Living Atlas website.

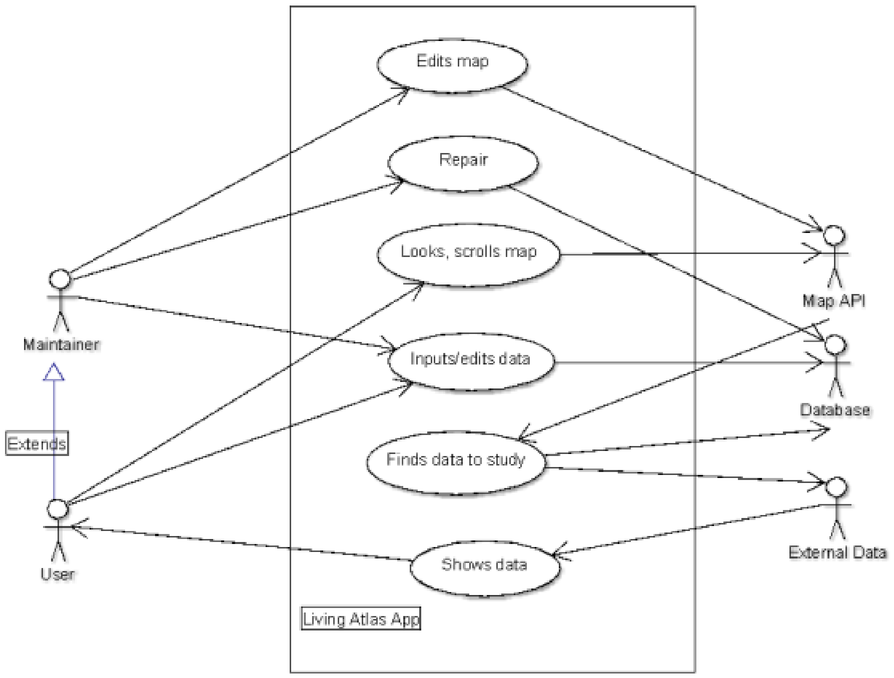
Sierra Svetlik is a computer science student who loves to code. She is currently pursuing a Bachelor of Computer Science at Washington State University. She has experience with C/C++, Python, and SQL. Sierra also has some experience with C#, Haskell, and JavaScript. She has a goal of developing video games but would be happy to apply her knowledge anywhere to make a positive impact on the world. She is part of Living Atlas team 1.

III. Project Requirements

This document serves as an update of CEREO Living Atlas's progress on the Living Atlas app. It will cover the current progress of CLA on the Living Atlas app. Included in this document will be details on the project and team members, Living Atlas specifications and requirements, our approach to solving the given the issue, and the plans for testing the app. This document will also provide updates of the project and details of the progress made.

III.1. Use Cases

CLA has been tasked with making an app that will be known as Living Atlas. The Living Atlas app will be able to collect environmental information about the Columbia River Basin of many different types, as well as having this information accessible by anyone. The ability to input information in the app will be restricted, but the ability to view the information will not be. The target groups for the app includes researchers, locals, tribes, and government officials. The end goal of this project is to have a repository of a wide variety of environmental data that can be accessed by anyone and uploaded and edited by anyone with permission.



Story: A student wants to write a paper on nitrogen in the Columbia River Basin. To find information about the presence of nitrogen in the river, the student goes to the Living Atlas application. After looking around the map a bit, they find a spot of interest and look up nitrogen using either a search bar feature or a category menu. Papers and other data related to the search or category appear in a menu next to the map, these results being tied to the area of the map that the student is looking at. The results were found using categories tagged to them or finding the word "nitrogen" in their title. The student is then able to open the papers or other data to use in their research, and can be confident in their validity, as only verified users and data can be uploaded to the application.

Source: CEREO Staff Member

Story: A researcher is studying the clusters of salmon in the Columbia River Basin. After collecting numbers for different clusters of salmon throughout a section of the river, they want to upload it to the Living Atlas. They first log into the application, then pinpoint an area where they found one of the clusters of salmon. They choose to upload data to that coordinate, start filling in a template for their data, including various different kinds of information about their data. This info includes their name, data of collection/upload, among other types of info (to be determined). They then upload their research. Next, the file is given an ID so that it can be uniquely recognized inside the database and is tagged with various categories that it can be found under. This process is repeated for all the different coordinates that the researcher found cluster of salmon at.

Source: CEREO Staff Member

III.2. Functional Requirements

1. Data collection:

- The system must provide a user-friendly interface for collecting and uploading environmental data from various sources, including individuals, institutions, and government agencies.
- The system must verify the accuracy and validity of the uploaded data through a verification process.
- The system must provide feedback to the uploader in case of invalid or inaccurate data.

Source: Environmental experts working on the Living Atlas project originated this requirement. The requirement is necessary for collecting accurate and reliable environmental data.

Priority: Priority Level 0: Essential and required functionality

2. Data management:

- The system must provide efficient and organized storage for the collected data, allowing easy accessibility and searchability.
- The system must support sorting, filtering, and analysis of the data to gain valuable insights.
- The system must support data versioning to keep track of changes and revisions made to the data over time.

Source: Environmental experts working on the Living Atlas project originated this requirement. The requirement is necessary for effectively managing and utilizing the collected environmental data.

Priority: Priority Level 0: Essential and required functionality

3. Data sharing:

- The platform must provide an interface for sharing environmental data with the wider community, promoting collaboration, and improving access to data.
- The platform must support secure and controlled sharing of data with specific individuals or groups.
- The platform must provide a notification system to inform users of new or updated data.

Source: Environmental experts working on the Living Atlas project originated this requirement. The requirement is necessary for promoting collaboration and providing valuable insights into the environment.

Priority: Priority Level 0: Essential and required functionality

4. External data connection:

- The system must support connections to external data sources to expand the range of environmental data available on the platform.
- The system must support various data formats and protocols for importing data from external sources.

Source: Environmental experts working on the Living Atlas project originated this requirement. The requirement is necessary for providing a comprehensive view of the environment and enabling users to gain valuable insights.

Priority: Priority Level 1: Desirable functionality

5. Data visualization:

- The system must provide an interactive map-based interface for visualizing environmental data in a geographical context.
- The map-based interface must support various map layers and overlay options for viewing data in different ways.
- The interface must allow users to filter and customize the displayed data based on various parameters.

Source: Environmental experts working on the Living Atlas project originated this requirement. The requirement is necessary for providing a comprehensive view of the environment and enabling users to gain valuable insights.

Priority: Priority Level 0: Essential and required functionality

III.3. Non-Functional Requirement

Performance: The Living Atlas website should load quickly and respond to user interactions promptly, even during peak traffic periods. The site should also be scalable and able to handle increases in traffic without slowing down.

Usability: The site's user interface should be intuitive and easy to navigate, with a clear hierarchy of information and consistent design elements throughout the site. The site should also be accessible to users with disabilities, with features like alt text for images and keyboard navigation. It should also be inclusive for users who are apart of native tribes in the pacific northwest region.

Security: The Living Atlas website should protect users' personal information, including passwords, uploaded/linked files, and personal data. The site should use encryption and other security measures to prevent unauthorized access and data breaches.

Compatibility: The site should be compatible with a wide range of browsers, devices, and operating systems, so that users can access it from wherever they are and on whatever device they choose.

Reliability: The Living Atlas website should be available and responsive at all times, with mechanisms in place to handle outages and downtime. The site should also have backup and recovery measures in place to ensure that data is not lost in case of a failure.

Maintainability: Once the Living Atlas website is deployed it will be maintained by WSU technical department. Therefore, it should be built with maintainability in mind, with clean and well-documented code, and should be easy to modify or update as needed.

Efficiency: The Living Atlas website should be optimized for performance and efficency to reduce page load times and minimize server load. The site should also be designed with energy efficiency in mind, using best practices for web development to minimize energy consumption.

Data Management: The site should be able to manage large volumes of data, such as information relating to watersheds, rivers, and communities, with high availability, scalability, and reliability. It should also have robust data backup and recovery mechanisms in place.

Searchability: The site's search functionality should be efficient, accurate, and fast. It should return relevant results, provide filtering options, and be able to handle large volumes of data.

IV. Software Design – From Solution Approach

IV.1. Front-End

Subsystem Decomposition:

Header Component:

1) Description: The Header component displays the logo and navigation links, Search bar and filtering button, and Upload Button.
2) Concepts and Algorithms Generated: The Header component utilizes concepts such as UI design and user experience, as well as algorithms for rendering and positioning UI elements.

Upload Form Component:

1) Description: The Upload Form component provides users with a way to upload data to the application, including selecting a file to upload and entering metadata about the data. The Form must have eleven variables, name, email, funding, organization, title, link, description, tags, latitude, longitude, and file.

2) Concepts and Algorithms Generated: The Upload Form component utilizes concepts such as form validation and data modeling, as well as algorithms for handling file uploads and data storage. Also, it must have the ability to send a POST request to the Backend.

Content Area Component:

1) Description: The Content Area component is responsible for fetching data from the backend using a GET request and rendering the Card components based on the data retrieved.

2) Concepts and Algorithms Generated: The Content Area component utilizes concepts such as asynchronous programming and data fetching, as well as algorithms for parsing and processing data retrieved from the backend.

Card Component:

1) Description: The Card component displays information about a specific piece of data, including name, email, funding, organization, title, link, description, tags, latitude, longitude, and file.

2) Concepts and Algorithms Generated: The Card component utilizes concepts such as data modeling and UI design, as well as algorithms for rendering UI elements based on data passed in.

Front-End Software Design:

This section describes the Front-End Software Design. First, this section will outline the foundational design philosophy. Then, this section will explain how each subcomponent interacts to form a cohesive application.

The Front-End Software design for Living Atlas was inspired by the real estate website Zillow, aiming to harness their intuitive user experience and functionality. However, it was equally important for us to carve out our own design. Therefore, while drawing inspiration we strived to introduce originality, ensuring our platform wouldn’t be a mere replica. Simplicity was another important variable for our approach. We wanted to create a user interface that, even with its unique elements, remained easy to navigate and understand.

The Front-End is organized into several main components. The Header Component displays the logo, navigation links, a search bar with a filter button, and an upload button. Next, the Upload Form Component, which is located inside the Header component, lets users add data to the app. This form asks for details like name, email, and more. It uses basic checks and flow diagram to ensure data is correct and can send data to the backend. The Content Area Component gets data from the backend and shows its data. One of the Content Area Component will use the Card Component. This component displays individual data items like name, email, and other details. Inside this card component there will be a download button that will download any type of file.

IV.2. Back-End

The Backend is made up of restful endpoints that allow the frontend to access the data from the database in a clean and concise matter. All endpoints in this project are made using FastAPI with Python.

High-Level Back-End Design:

This section describes Back End Software Design. I will outline the foundational design philosophy. Then, I will explain how each endpoint interacts with the frontend and database to form a cohesive application.

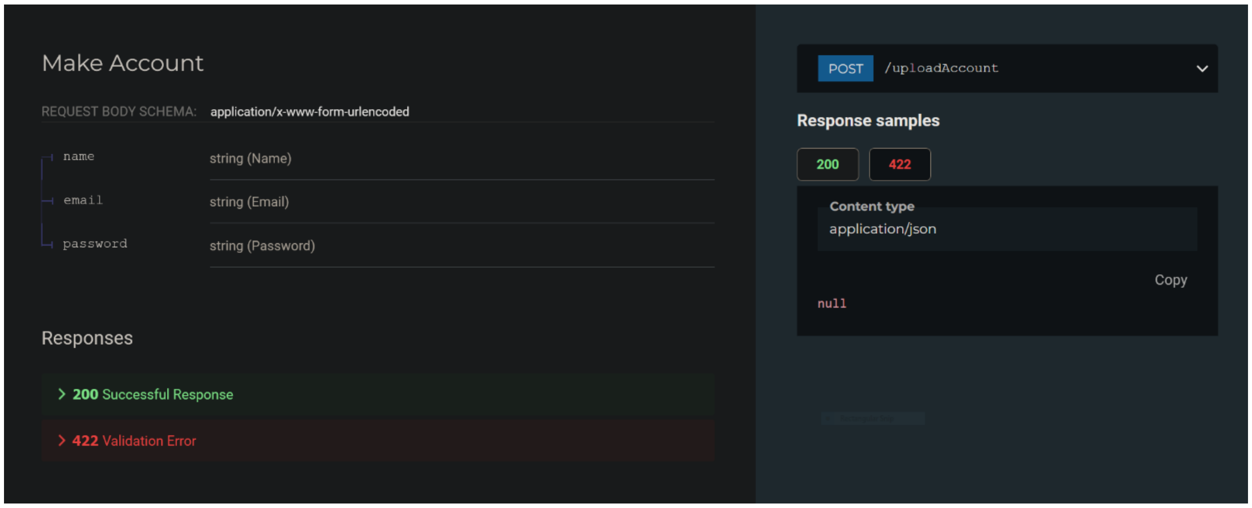
The Back-End Software design for Living Atlas has four foundational features. The sign in, filters, map, and card endpoints. The sign in endpoint functionally takes care of the creation, authentication, and verification of user accounts and determining the status the logged in user account. The filter endpoint functionality takes care of querying the database to retrieve certain cards on the right side of the website. Filters are customizable and therefore the filter options will vary based on the content of the site. The map endpoint functionality takes care of supplying the living atlas map team with the necessary data to visually show the card items in their correct location and to filter the card items if the user creates a box around a particular area in the map. The card endpoint functionality takes care of retrieving data of the set of cards when the site is initially loaded and creating/deleting cards.

Component-Level Endpoint Decomposition:

- *All screenshots provided in this section are from the REST API documentation tool Swagger UI. Specifically, the ".../redoc" and ".../docs" sections for better visualization.*

Register Account Endpoint:

- Description: This endpoint is a POST request from the frontend that is for when the user wants to create a new account.


Make Account

REQUEST BODY SCHEMA: application/x-www-form-urlencoded

name	string (Name)
email	string (Email)
password	string (Password)

Responses

> 200 Successful Response

> 422 Validation Error

POST /uploadAccount

Response samples

200 422

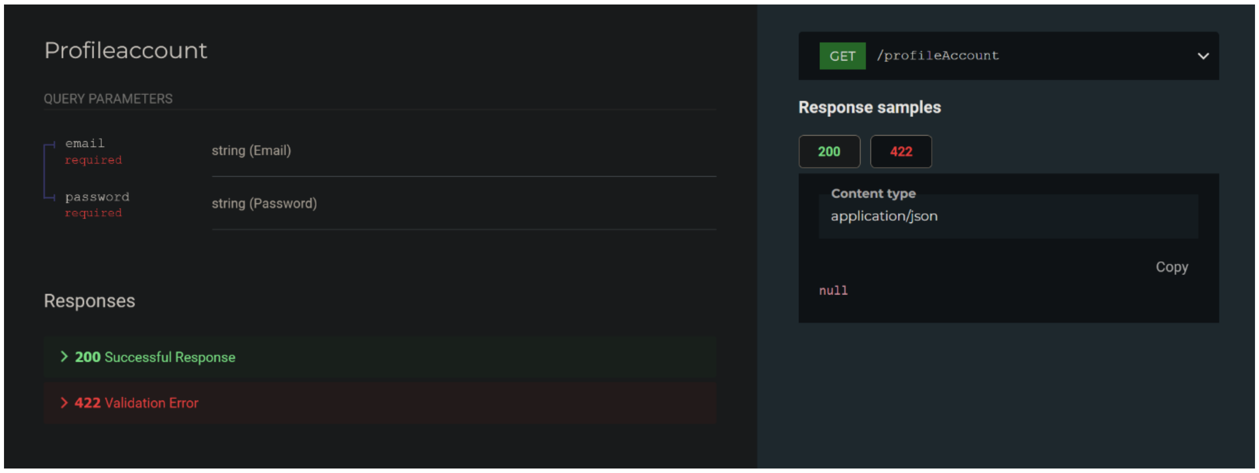
Content type
application/json

Copy

```
null
```

Login Endpoint:

- Description: This endpoint is a GET request and will be used when the user clicks on the login button and enters in their account information. This endpoint verifies that the user submitted information is correct.


Profileaccount

QUERY PARAMETERS

email required	string (Email)
password required	string (Password)

Responses

> 200 Successful Response

> 422 Validation Error

GET /profileAccount

Response samples

200 422

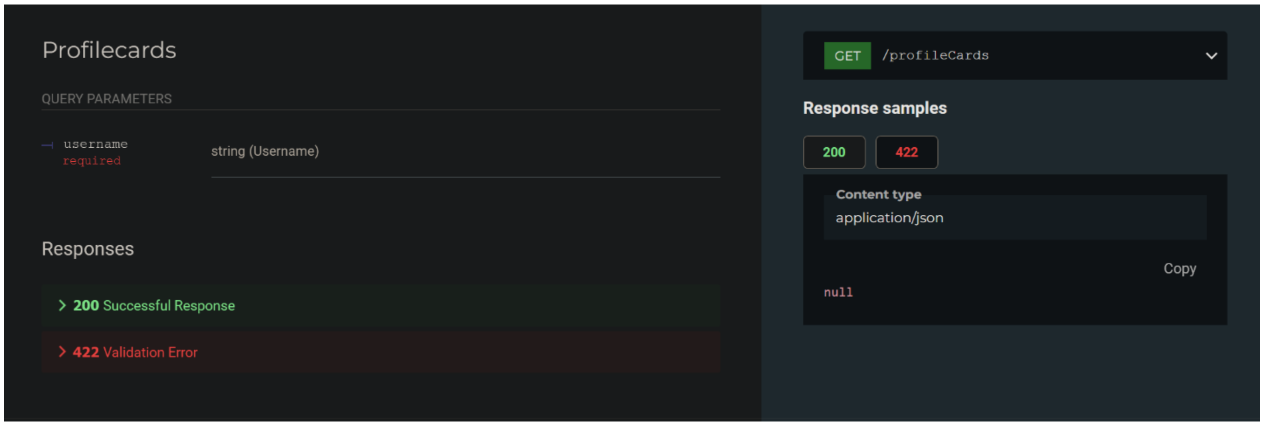
Content type
application/json

Copy

```
null
```

Profile Cards Endpoint:

- Description: This endpoint is a GET request and is used when the user clicks on the account button on the front end. The endpoint will take in a unique username (sent by the front end because the user is logged in) and return to the front end all the cards that this particular user has uploaded.


Profilecards

QUERY PARAMETERS

username required	string (Username)

Responses

> 200 Successful Response

> 422 Validation Error

GET /profileCards

Response samples

200 422

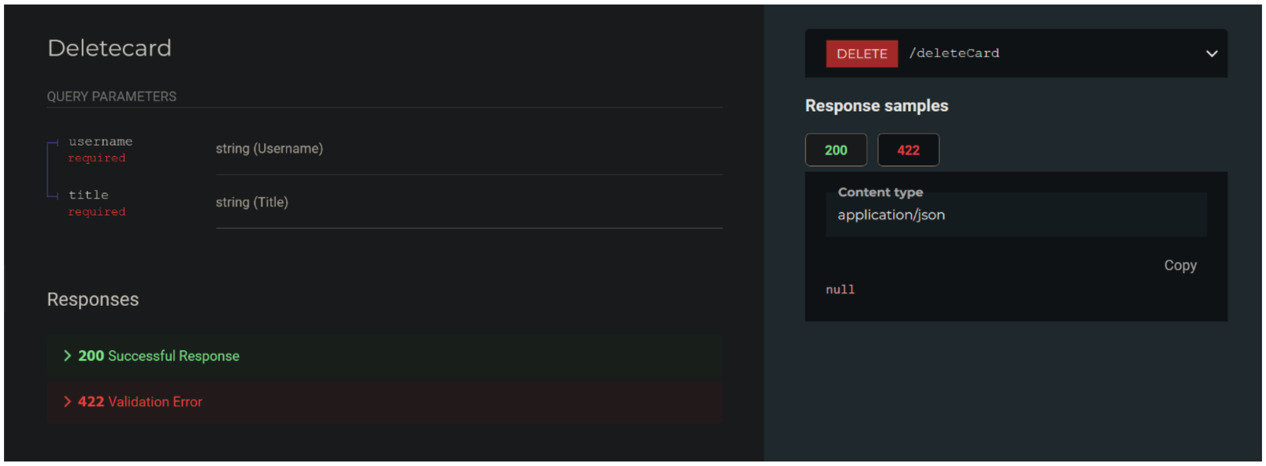
Content type
application/json

Copy

```
null
```

Delete Card Endpoint:

- Description: This endpoint is a DELETE request and is used when a user is on the account page of the site and viewing the cards they have uploaded. This endpoint takes in a username and a card title to delete cards because titles can duplicate.


Deletecard

QUERY PARAMETERS

username required	string (Username)
title required	string (Title)

Responses

> 200 Successful Response

> 422 Validation Error

DELETE /deleteCard

Response samples

200 422

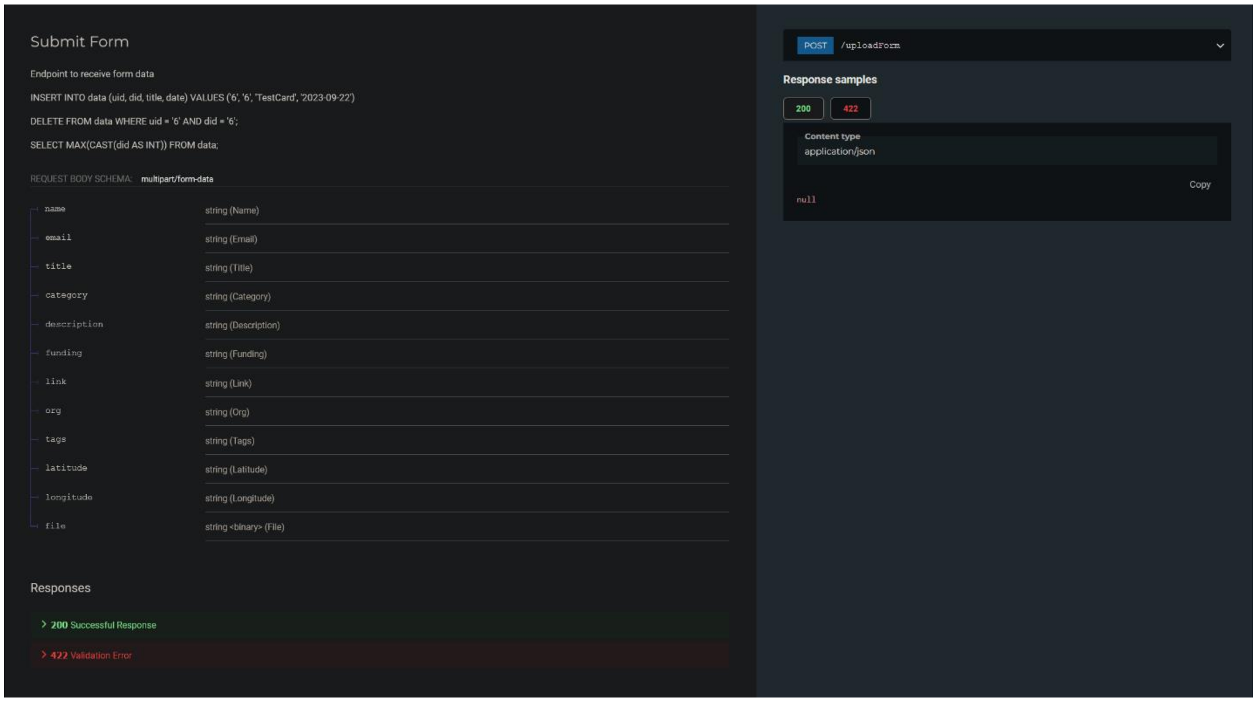
Content type
application/json

Copy

```
null
```

Upload Card Endpoint:

- Description: This endpoint is a POST request and will be used when the user wants to create their own card data point to be displayed on our site. Card data points include this data that will be sent over in a form from the front end to the back end. Username, email, title, category, description, funding, link, organization, all tags, latitude, longitude, and a file. After that the backend verifies the legitimacy of the data and if it passes it will query the database using INSERT to input the data appropriately.



Submit Form

Endpoint to receive form data

INSERT INTO data (uid, did, title, date) VALUES ('6', '6', 'TestCard', '2023-09-22')

DELETE FROM data WHERE uid = '6' AND did = '6';

SELECT MAX(CAST(did AS INT)) FROM data;

REQUEST BODY SCHEMA: multipart/form-data

`name`	string (Name)
`email`	string (Email)
`title`	string (Title)
`category`	string (Category)
`description`	string (Description)
`funding`	string (Funding)
`link`	string (Link)
`org`	string (Org)
`tags`	string (Tags)
`latitude`	string (Latitude)
`longitude`	string (Longitude)
`file`	string <binary> (File)

Responses

> 200 Successful Response

> 422 Validation Error

POST /uploadForm

Response samples

200 422

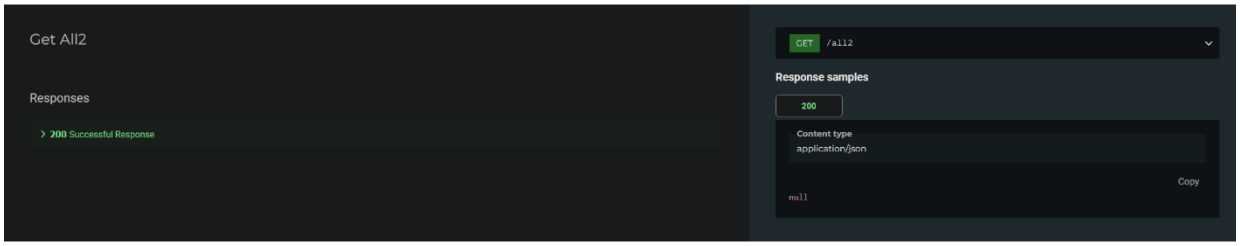
Content type
application/json

Copy

```
null
```

Retrieve All Cards Endpoint:

- Description: This endpoint is a GET request and is used when the Living Atlas is initially loaded or whenever it is refreshed. It returns the last 12 cards that were submitted to populate the content of the card list in the Atlas.



Get All2

Responses

> 200 Successful Response

GET /all2

Response samples

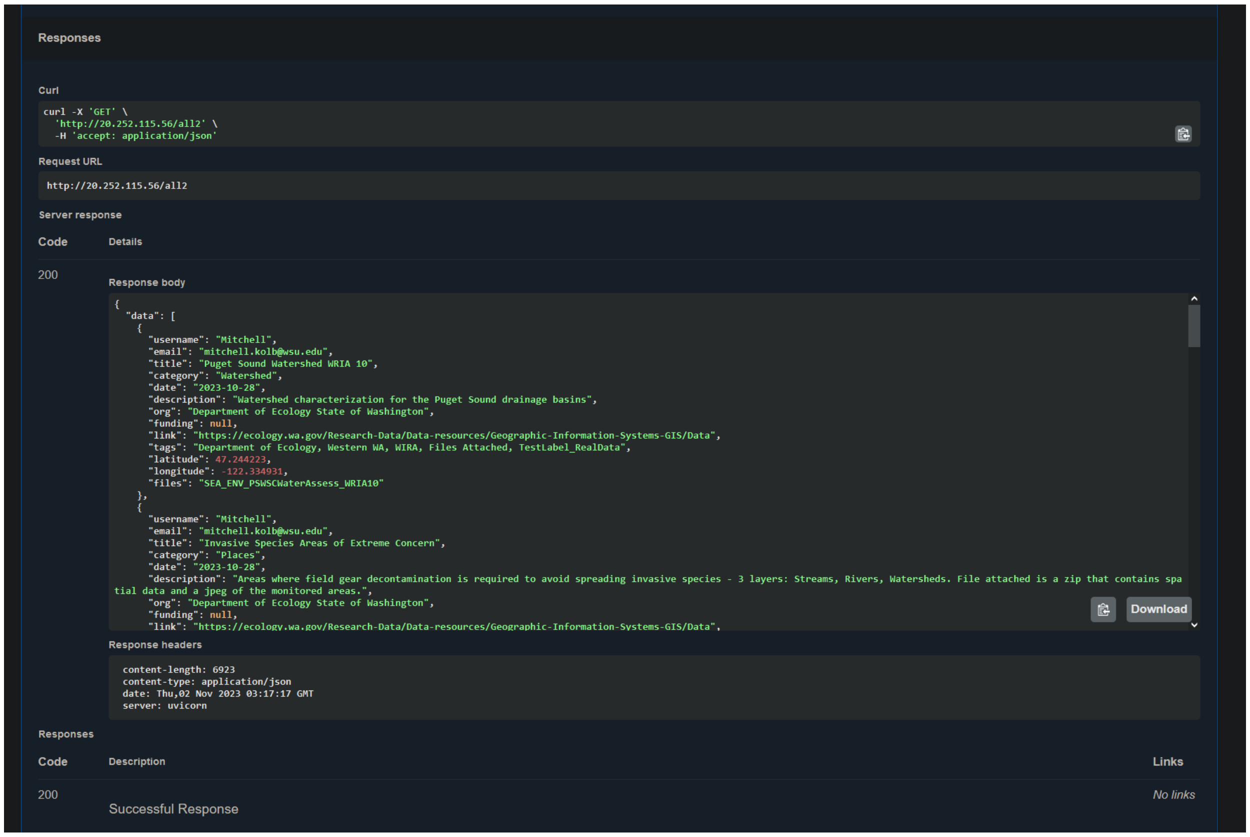
200

Content type
application/json

Copy

```
null
```

Example Output


Responses

Curl

```
curl -X 'GET' \
  'http://20.252.115.56/all2' \
  -H 'accept: application/json'
```

Request URL

```
http://20.252.115.56/all2
```

Server response

Code Details

200

Response body

```
{
  "data": [
    {
      "username": "Mitchell",
      "email": "mitchell.kolb@wsu.edu",
      "title": "Puget Sound Watershed WRIA 10",
      "category": "Watershed",
      "date": "2023-10-28",
      "description": "Watershed characterization for the Puget Sound drainage basins",
      "org": "Department of Ecology State of Washington",
      "funding": null,
      "link": "https://ecology.wa.gov/Research-Data/Data-resources/Geographic-Information-Systems-GIS/Data",
      "tags": "Department of Ecology, Western WA, WIRA, Files Attached, TestLabel_RealData",
      "latitude": 47.244223,
      "longitude": -122.334931,
      "files": "SEA_ENV_PSWSCWaterAssess_WRIA10"
    },
    {
      "username": "Mitchell",
      "email": "mitchell.kolb@wsu.edu",
      "title": "Invasive Species Areas of Extreme Concern",
      "category": "Places",
      "date": "2023-10-28",
      "description": "Areas where field gear decontamination is required to avoid spreading invasive species - 3 layers: Streams, Rivers, Watersheds. File attached is a zip that contains spa
tial data and a jpeg of the monitored areas.",
      "org": "Department of Ecology State of Washington",
      "funding": null,
      "link": "https://ecology.wa.gov/Research-Data/Data-resources/Geographic-Information-Systems-GIS/Data",
```

Response headers

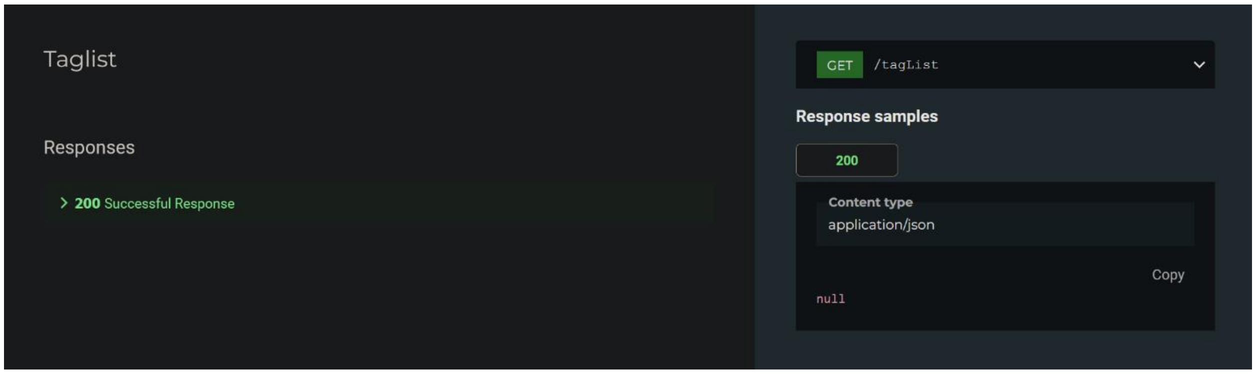
```
content-length: 6923
content-type: application/json
date: Thu,02 Nov 2023 03:17:17 GMT
server: uvicorn
```

Responses

Code	Description	Links
200	Successful Response	*No links*

Custom Tag List Endpoint:

- Description: This endpoint is a GET request and is intended to be used to populate a drop-down menu for filtering through the custom tags. It only returns the custom tags for submitted cards that are unique with no duplicates.


Taglist

Responses

> 200 Successful Response

GET /tagList

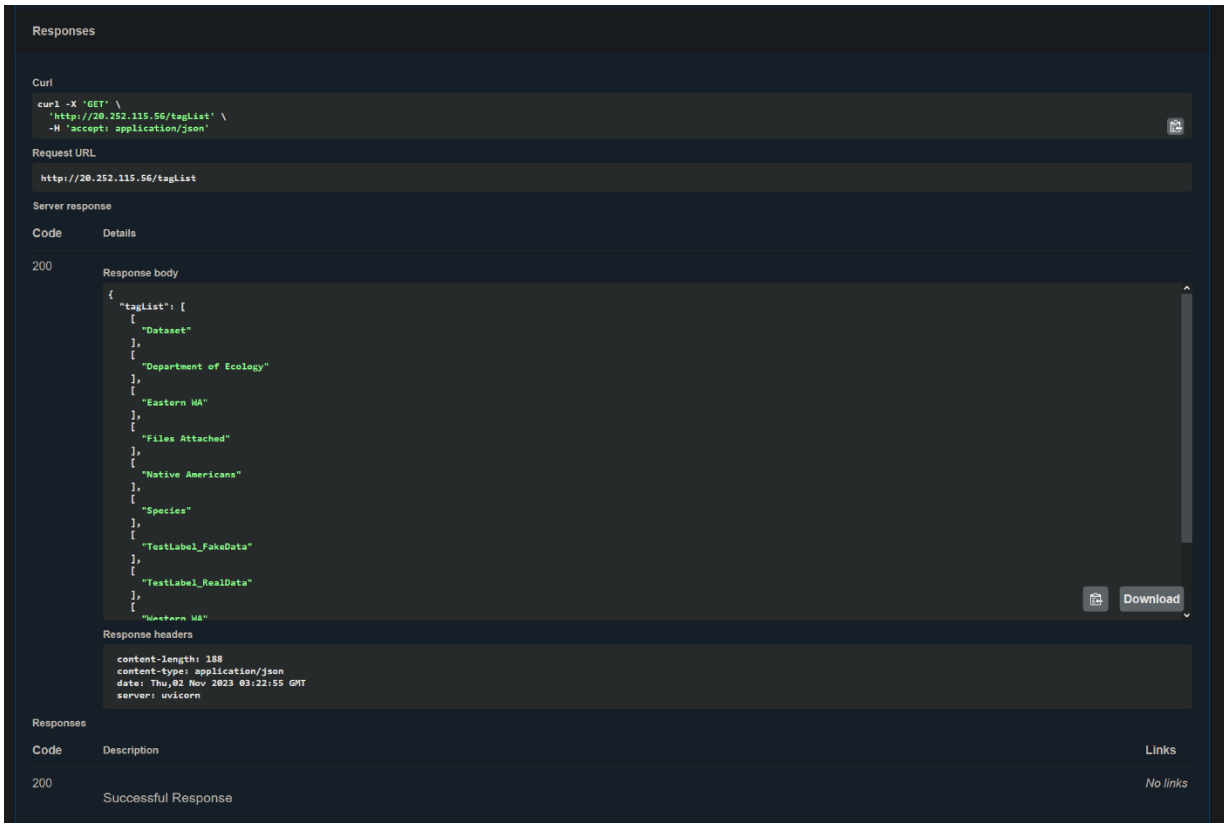
Response samples

200

Content type
application/json

```
null
```

Example Output


Responses

Curl

```
curl -X 'GET' \
  'http://20.252.115.56/tagList' \
  -H 'accept: application/json'
```

Request URL

```
http://20.252.115.56/tagList
```

Server response

Code	Details
200	Response body

```
{
  "tagList": [
    [
      "Dataset"
    ],
    [
      "Department of Ecology"
    ],
    [
      "Eastern WA"
    ],
    [
      "Files Attached"
    ],
    [
      "Native Americans"
    ],
    [
      "Species"
    ],
    [
      "TestLabel_FakeData"
    ],
    [
      "TestLabel_RealData"
    ],
    [
      "Western WA"
```

Response headers

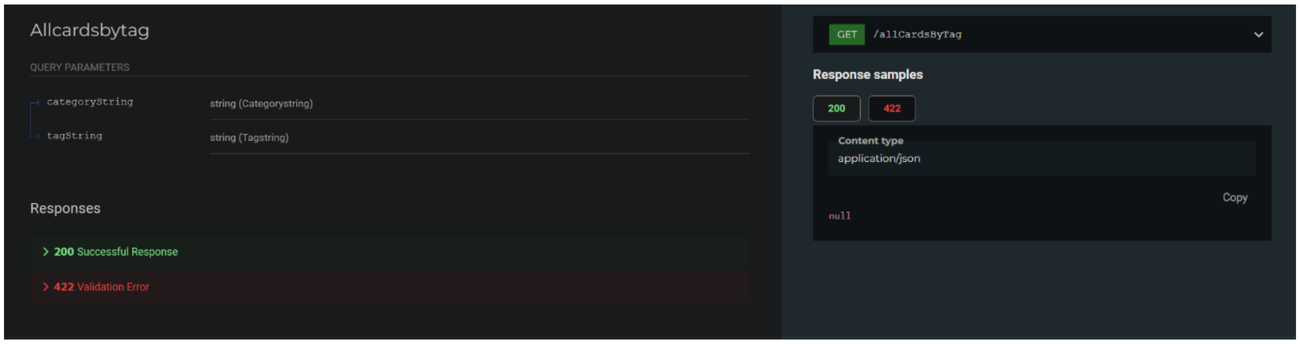
```
content-length: 188
content-type: application/json
date: Thu,02 Nov 2023 03:22:55 GMT
server: uvicorn
```

Responses

Code	Description	Links
200	Successful Response	No links

Filter Cards by Custom Tag Endpoint:

- Description: This endpoint is a GET request and is used to filter the card list when the user adjusts the category and/or custom tag filters on the filter bar.


Allcardsbytag

QUERY PARAMETERS

categoryString	string (Categorystring)
tagString	string (Tagstring)

Responses

> 200 Successful Response

> 422 Validation Error

GET /allCardsByTag

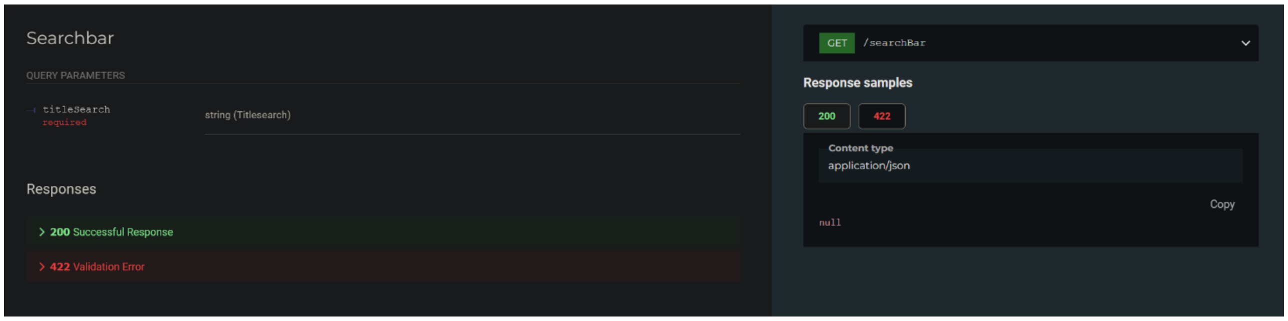
Response samples

Content type
application/json

```
null
```

Filter Cards by Title Endpoint:

- Description: This endpoint is a GET request and is used to filter the card list when the user enters in a card title filter on the filter bar. This endpoint is only sensitive for title spelling. Casing doesn't affect the results.



Searchbar

QUERY PARAMETERS

titleSearch required — string (Titlesearch)

Responses

> 200 Successful Response

> 422 Validation Error

GET /searchBar

Response samples

200 422

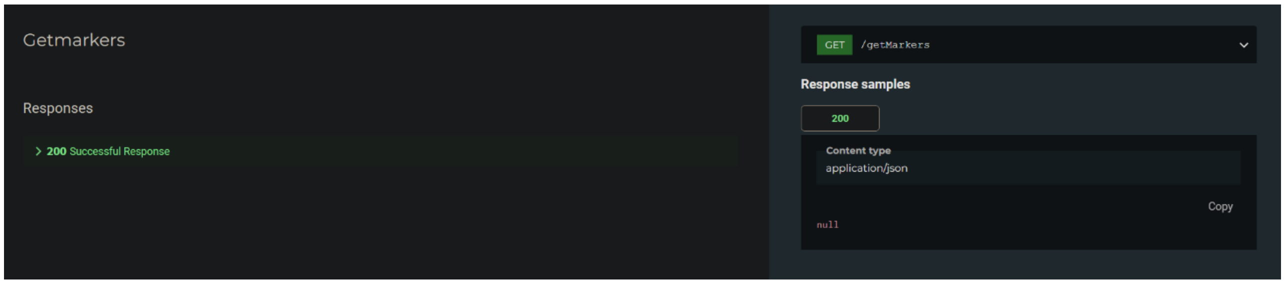
Content type
application/json

Copy

```
null
```

Populate Map Point Locations Endpoint:

- Description: This endpoint is a GET request and is used by the map to retrieve Latitude and Longitude and card title data from the cards that are being displayed.



Getmarkers

Responses

> 200 Successful Response

GET /getMarkers

Response samples

200

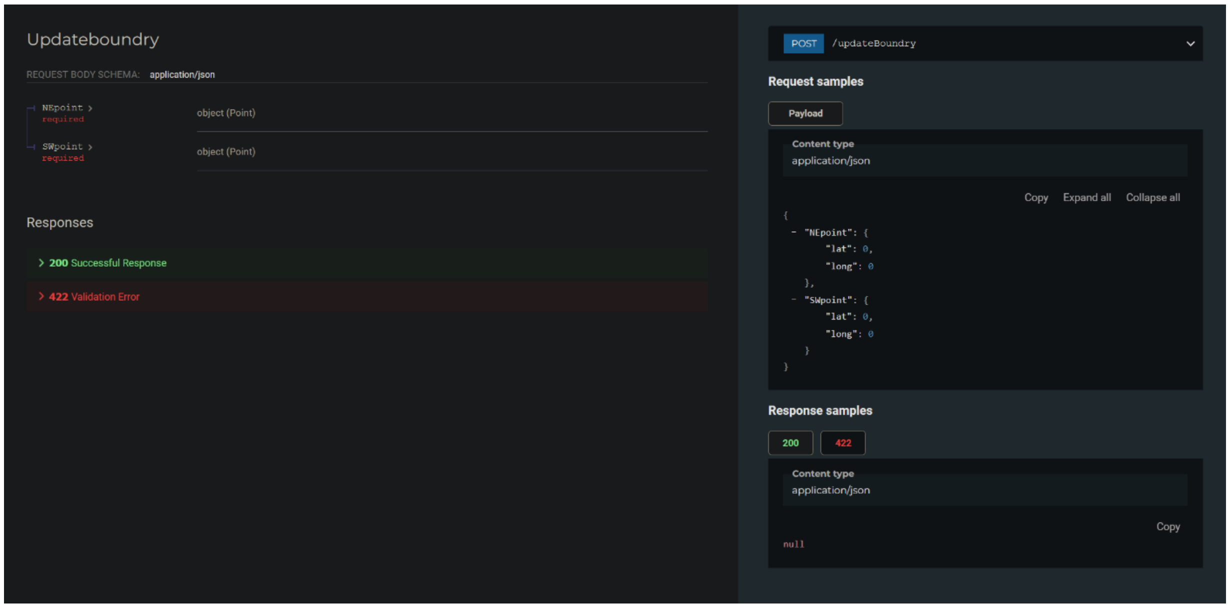
Content type
application/json

Copy

```
null
```

Update Card List Based on Map Condition Endpoint:

- Description: This endpoint is a POST request and is used when the user pans or zooms on the map. It will retrieve the cards that are in the specified Lat/Long square that the user map currently views.



Updateboundry

REQUEST BODY SCHEMA: application/json

NEpoint > required — object (Point)

SWpoint > required — object (Point)

Responses

> 200 Successful Response

> 422 Validation Error

POST /updateBoundry

Request samples

Payload

Content type
application/json

Copy Expand all Collapse all

```
{
  - "NEpoint": {
        "lat": 0,
        "long": 0
    },
  - "SWpoint": {
        "lat": 0,
        "long": 0
    }
}
```

Response samples

200 422

Content type
application/json

Copy

```
null
```

IV.3. Database

The database will utilize various tables to store the information efficiently. The primary key is the smallest amount of information needed to uniquely identify the row the data is stored in. The foreign key is the information that can be utilized to connect the information in one table with the information in another table.

Note: In PostgreSQL, string is known as character varying

ER Relationship Diagram:

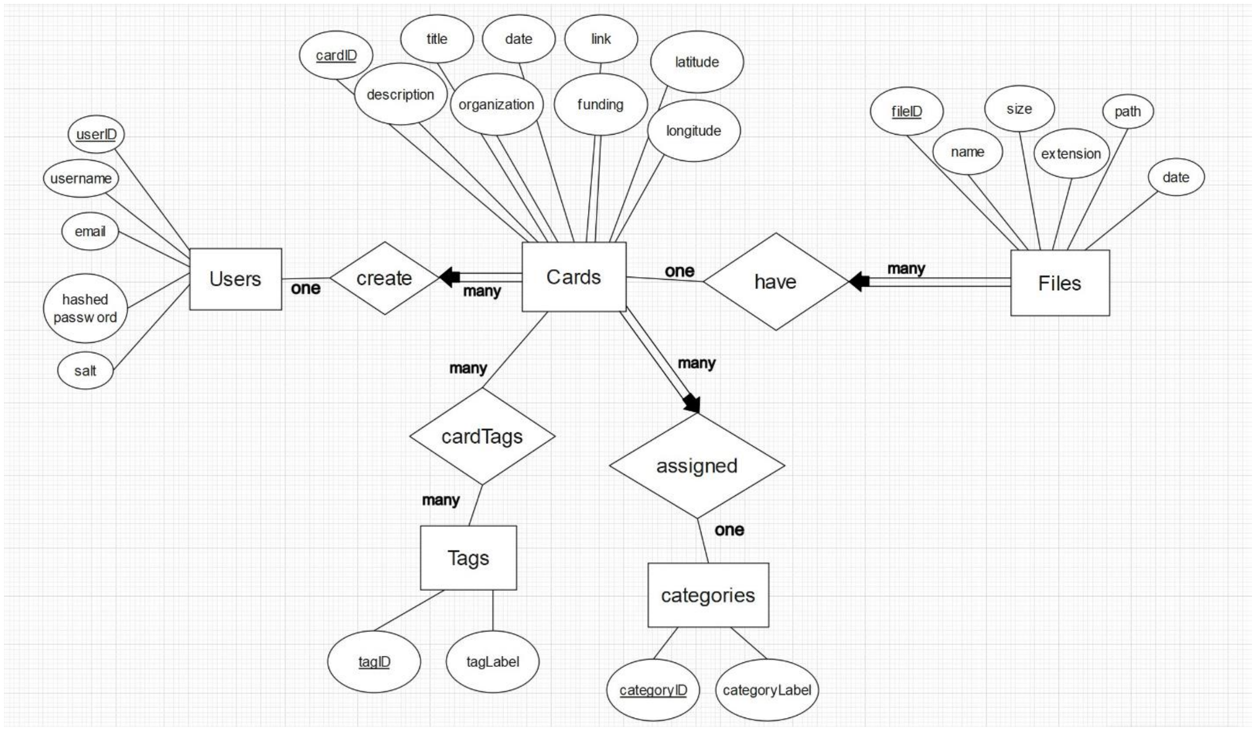


Table Schema:

```
CREATE TABLE Users (
    UserID INT,
    Username VARCHAR(255) NOT NULL,
    Email VARCHAR(255) UNIQUE NOT NULL,
    HashedPassword VARCHAR(255) NOT NULL,
    Salt BYTEA NOT NULL,
    PRIMARY KEY (UserID)
```

```
);

CREATE TABLE Categories (
    CategoryID INT,
    CategoryLabel VARCHAR(255) UNIQUE,
    PRIMARY KEY (CategoryID)
);

CREATE TABLE Cards (
    CardID INT,
    UserID INT NOT NULL,
    Title VARCHAR(255) NOT NULL,
    Latitude DECIMAL(10,8) NOT NULL,
    Longitude DECIMAL(11,8) NOT NULL,
    CategoryID INT NOT NULL,
    DatePosted DATE DEFAULT CURRENT_DATE,
    Description VARCHAR(2000),
    Organization VARCHAR(255),
    Funding DECIMAL(10,2),
    Link VARCHAR(255),
    PRIMARY KEY (CardID),
    FOREIGN KEY (UserID) REFERENCES Users(UserID),
    FOREIGN KEY (CategoryID) REFERENCES Categories(CategoryID)
);

CREATE TABLE Files (
    FileID INT,
    CardID INT NOT NULL,
    FileName VARCHAR(255),
    DirectoryPath VARCHAR(255),
    FileSize INT, --Store all file sizes in the type of bytes
    FileExtension VARCHAR(20),
    DateSubmitted DATE DEFAULT CURRENT_DATE,
    PRIMARY KEY (FileID),
    FOREIGN KEY (CardID) REFERENCES Cards(CardID)
);

CREATE TABLE Tags (
    TagID INT,
    TagLabel VARCHAR(255),
    PRIMARY KEY (TagID)
);

CREATE TABLE CardTags (
    CardID INT REFERENCES Cards(CardID),
    TagID INT REFERENCES Tags(TagID),
    PRIMARY KEY (CardID, TagID),
    FOREIGN KEY (CardID) REFERENCES Cards(CardID),
    FOREIGN KEY (TagID) REFERENCES Tags(TagID)
);
```

Schema breakdown:

Users table:

The users table will store information about the users of the app, with primary key userid, which is unique to every user. The userid is an integer. The users table will also store the username (string), the password (known as hashedpassword) in a hashed form (string), the email (string), and the salt for the password (bytea).

The password being stored in a hashed form allows for the user's password to be stored safely without compromising the user's security. When the user types their password, the password will be hashed, the salt will be added on, as well as a pepper stored in the code. The combination of the password, salt, and pepper then gets hashed, and is compared to the hashed password in the database. If that matches, the user is granted entry into the app. Hashing the password allows for safety because the hash cannot be undone.

Categories table:

The categories table will store information related to categories for the data, with primary key categoryid, which is unique to every category. The categoryid is an int. The categories table also stores the categorylabel (string) which is the name of the category.

Cards table:

The cards table will store some of the information about the data uploaded to the app, with primary key cardid and foreign keys userid and categoryid. The cardid is an int, unique to every piece of data uploaded. The userid will allow for access to information about the user that uploaded the data without storing that information in the cards table. The categoryid will allow for storing the category of the data, and the category itself is accessed from the categories table. The cards table will also store the title of the data (string), the latitude and longitude (decimal) that the data corresponds to on the map, the date the data was uploaded, known as dateposted (date), a description (string) of the data, organization (string) that the user is a part of (if they are and choose to share it), the source (string) of funding (should the user have one and chooses to share it), and a link (string) to the source of the data (if it is from an outside source) or to a separate website (if the user chooses to link to one).

The table being known as the cards table refers to how the data is represented in the app using cards shown next to the map.

Files table:

The files table will store some of the information about files uploaded to the server (if the files are not sourced from a separate website), with primary key fileid and foreign key cardid. The fileid is an int, unique to every file uploaded. The cardid will allow access to information about the file that is not stored in the files table. The files table will also store the name of the file (string), known as filename, the path of the file will be stored in in the server (string), known as directorypath, the size of the file (string), known as filesize, the extension of the file (string), known as fileextension, and the date the file the was uploaded (date), known as datesubmitted.

Tags table:

The tags table stores information about the tags, with primary key tagid, which are ints. The tags table also stores the names of the tags (strings), known as taglabel.

Storing the names of the tags separately allows for easy retrieval of the names of the tags, especially since the data can have multiple tags.

Cardtags table:

The cardtags table stores information about the data to associate with the tags, with pairs of cardids and tagids acting as the primary key, and both the cardid and tagid acting as foreign keys. The cardids might repeat throughout the table, and so can the tagids, so the tuples of the cardid and tagid act as the primary key.

Explanation:

This schema works best for the app because the information is stored as efficiently as possible, allowing for minimal repetition of information, yet still allowing for all the information to be retrievable. The users table can be reached from the cards table. The cards table can be retrieved from the file table, and the cardtags table. The tags table can be retrieved from the cardtags table. And the categories table can be retrieved from the cards table.

V. Test Plan

V.1. Overview

Test Objectives and Schedule:

The test objectives for the Living Atlas application include testing the primary functions of the application. The testing schedule includes conducting functional testing during the development phase and conducting acceptance testing with sponsors and clients during the staging phase. Additionally, non-functional requirements such as performance testing will be conducted using appropriate testing strategies and tools. The testing process will be iterative, with bugs and issues being reported and addressed throughout the development and staging phases. Overall, the testing objectives and schedule aim to ensure the quality and functionality of the Living Atlas application.

Testing strategies and tools for functional requirement:

For testing the functional requirements of our Living Atlas application, we will be using a combination of manual and automated testing. Manual testing will involve testing each individual component and feature of the application to ensure they are functioning correctly. Automated testing will be carried out using Selenium for the front end, which will allow for efficient and accurate testing of the application's functionality across different browsers and platforms. Postman will be used for the backend to test accuracy of the returned data.

Testing strategies and tools for non-functional requirement:

For testing non-functional requirements such as performance and security, we will be using tools such as JMeter, Loadster. These tools will allow us to simulate real-world scenarios and identify any bottlenecks or vulnerabilities in the application. Additionally, the front end will be setting up alerts and monitoring tools to keep track of the application's performance and detect any issues in real-time. The back end will be using a program called Locust which can simulate user behavior and test the performance of the FastAPI endpoints. It can do this by being able to generate thousands of concurrent users and measure the response time throughout the API. For specific endpoint testing the backend will also take advantage of FastAPI's built-in testClient functionality to test if input and output and response codes are correct.

V.2. Front-End

Functional Testing:

For the functional testing of our Living Atlas application, we will be focusing on testing the primary/core functions of the application. For the Front-End this includes testing the ability to upload and download data, search for data by keyword, displaying data by cards. We will be using the following test cases to ensure that the core functions of the application are working as expected.

1. Test the ability to upload data to the Living Atlas application

Steps	Action	Expected Response	Pass/Fail	comments
1	Navigate to the upload page	Successfully navigate to the upload page when press upload button	P	
2	Select a file to upload	Successfully select a file to upload when press upload button	P	
3	Verify that the file is uploaded successfully	Successfully receive confirmation that the file was uploaded successfully.	P	

2. Test the ability to download data from the Living Atlas application

Steps	Action	Expected Response	Pass / Fail	Comments
1	Navigate to the download page	Press the card and make it show the download button	P	
2	Search for the desired data	Able to show the data description	P	
3	Select the data to download	Able to press the download button	P	
4	Verify that the data is downloaded successfully	Download the data	P	

3. Test the ability to search for data by keyword Steps:

Steps	Action	Expected Response	Pass/Fail	comments
1	Navigate to the search page	See the search bar	P	
2	Enter a keyword to search for	See the text you entered	P	
3	Verify that the search results are returned correctly	See the text you enter is correct	P	

4. Test the ability to displaying data by cards properly

Steps	Action	Expected Response	Pass/Fail	comments
1	Use a mock data set with known values to ensure the cards are rendered correctly	Store mock data in the database	P	
2	Open the application	Should be able to launch application	P	
3	Verify that all card information is	Cards should show all the	P	

	displayed accurately and in the correct format.	content of the moch data		

V.3. Back-End

Functional Testing:

For the functional testing of our Living Atlas application, we will be focusing on testing the primary/core functions of the application. For the Back End this includes testing the ability to take in form data from the front end and return the correct data relating to it and to return data from various requests.

1. Test the ability to upload data to the Living Atlas application database

Steps	Action	Expected Response	Pass/Fail	comments
1	Be given a form data type that contains all inputted data for the new research point	All data is assigned/converted to variables that can be used within the endpoint. "Converted Complete" will be printed to the terminal	P	
2	Perform POST request by querying the database with the INSERT statement	"Completed POST" will be printed to the terminal and returned to the front end. "Failed to insert" will be returned otherwise	P	

2. Test the ability to return data from the Living Atlas application database

Steps	Action	Expected Response	Pass/Fail	comments

1	Be given a title or userid from the frontend	The data is assigned/converted to variables that can be used within the endpoint. “Converted Complete” will be printed to the terminal	P	
2	Perform GET request by querying the database with the SELECT statement	“Completed GET” will be printed to the terminal and returned to the front end. “Failed to insert” will be returned otherwise	P	

3. Test the ability to delete data from the Living Atlas application database

Steps	Action	Expected Response	Pass/Fail	comments
1	Be given a title or userid from the frontend	The data is assigned/converted to variables that can be used within the endpoint. “Converted Complete” will be printed to the terminal	P	
2	Perform DELETE request by querying the database with the DELETE statement	“Completed DELETE” will be printed to the terminal and returned to the front end. “Failed to insert” will be returned otherwise	P	

Nonfunctional Testing:

For the nonfunctional testing of our Living Atlas application, we will be focusing on testing the primary/core functions of the application. For the Back End this includes testing the performance of the endpoints to ensure that as this project scales up in the number of users it can still perform at the same efficiency and accuracy.

1. Test the ability to service more than one user who are using Living Atlas application

Steps	Action	Expected Response	Pass/Fail	comments
1	Async functionality on endpoints allows multiple users to request from the same endpoint.	Data is returned to both users		
2	Read and Writing to the local file server	Files can be found and created in the file server based off file paths		

VI. Test Case Specification & results

VI.1. Front-End

Front-End Code Based Test:

For the testing of my React application's codebase, I initially considered using Mockito. However, I quickly realized that Mockito is incompatible with React. Instead, I turned to the React Testing Library, paired with Jest. Using this combination, I was able to mock the main component and integrate it with the app. Despite these tools being well-suited for React, I encountered challenges in fully implementing codebase testing for React. I decide to focus more on specification testing.

Front-End Specification Based Test:

For the specification-based testing of our front-end components, I utilized Selenium, a powerful tool for controlling a web browser through programs and performing browser automation. The focus of my tests was on three critical components: the card display, filter, and erase functionalities. The card display test ensured that the visual elements, layout, and data were rendered correctly on the screen. For the filter component, I verified its ability to refine and present results based on given criteria, ensuring its responsiveness and accuracy. Lastly, the erase component testing ensured that the selected items or data points were correctly removed from the display without affecting the surrounding elements or leaving any residual data. Through these tests, I aimed to validate that each component met the specified requirements and functioned seamlessly in the user interface.

VI.2. Backend-End

Back-End Code Based Test:

For the testing of the FastAPI endpoints we only need to test endpoints that have input parameters. The GET request endpoints return the same structure of data every time unless the elephantSQL database service is down. If that is the case the backend will return "Database Service is Not Operational for every endpoint". All these tests are using testClient functions which are included with the FastAPI framework. They test response code and output results.

Test ID	Endpoint Name	Expected Inputs	Expected Outputs	Actual Outputs	Pass /Fail
1	/uploadAccount	Name=bob Email=bob@email.com Password=bobpass	{"success": True, "message": "New account added successfully"} Code:200	{"success": True, "message": "New account added successfully"} Code:200	P
2	/uploadAccount	Name=existingUser Email=existingUser@email.com Password=pass	{"success": False, "message": "All fields must be filled in"} Code:400	{"success": False, "message": "All fields must be filled in"} Code:400	P
3	/uploadAccount	Name= Email= Password=	{"success": False, "message": "All fields must be filled in"} Code:400	{"success": False, "message": "All fields must be filled in"} Code:400	P
4	/uploadAccount	Name=Josh Email=joshua.long@wsu.edu	{"success": False, "message": "Email must be unique"}	{"success": False, "message": "Email must be unique"}	P

		Password=josh	Code:400	Code:400	
5	/deleteCard	Username=Mitchell Title=TestCard	{"data": "Card TestCard Deleted"} Status:200	{"data": "Card TestCard Deleted"} Status:200	P
6	/deleteCard	Username= Title=	{"data": "Need more information"} Status:400	{"data": " Need more information "} Status:400	P
7	/deleteCard	Username=XZXZ Title=ZXZX	{"data": "Parameter 'name' is not in the database. Please make a new request to delete a card."} Status:400	{"data": "Parameter 'name' is not in the database. Please make a new request to delete a card."} Status:400	P
8	/uploadForm	Name=a Email=a Title=a Category=River Description= Funding= Link= Org= Tags= Lat=45.5 Long=112.6	{"success": True, "message": "File and Card Data Uploaded Successfully"} Status:200	{"success": True, "message": "File and Card Data Uploaded Successfully"} Status:200	P

9	/updateBoundry	47.245345
		-
		122.333112
		},
		"SWpoint": {
		"lat":
		47.243112,
		"long": -
		122.336764
		}
		}
			"Mitchell"
			"mitchell.kolb@wsu.edu"
			"Puget Sound Watershed WRIA 10"
				"Mitchell"
				"mitchell.kolb@wsu.edu"
				"Puget Sound Watershed WRIA 10"
					P

10	/updateBoundry	- 122.333112	{"data": Please make a new request to view a card."} Status:400	{"data": Please make a new request to view a card."} Status:400	P

		}, "SWpoint": { "lat": 47.243112, "long": -122.336764 } }			

VI.3. Database

While there aren't any specific tests that can be written for the database, ElephantSQL provides information about the performance of the database. However, the performance should not prove to be an issue as the database is not large, nor are the queries complex.

VII. Staging and Deployment

VII.1. Front-End

Staging: To stage the software for acceptance testing, we plan to create a separate staging environment that will be a replica of the production environment. This environment will be hosted on a separate server and will contain all the necessary infrastructure and dependencies required for the application to run.

Deployment: For the production deployment of the application, we plan to use the free tier of cloud-based platform, such as AWS or Google Cloud, to host the application.

Development and Deployment Environment: The development environment for the application is based on React, and we have set up a local development environment on our development machines. We use code editors like Visual Studio Code, and package managers like npm to manage dependencies and run the application locally. We use Git for version control, and we have set up a remote Git repository to manage the source code for the application. For the Deployment Environment we might have to use a Docker container or similar package that can be deployed to the production environment.

VII.2. Back-End

Staging: To stage a FastAPI application, we need to create a separate environment where the application can be tested and validated before deploying it to production. The staging environment should closely resemble the production environment, including the operating system, dependencies, and network configuration.

To set up a staging environment, we can use tools like Docker to create a containerized version of the application that can be easily deployed and tested on any machine. We can also use continuous integration and deployment (CI/CD) tools like Jenkins, TravisCI, or CircleCI to automate the deployment and testing process.

Deployment: When it comes to deploying a FastAPI application to production, we have several options depending on the specific needs and requirements of the project. Here are some common deployment strategies:

I. Cloud-based deployment: We can use cloud platforms like AWS, Google Cloud, or Azure to deploy the FastAPI application in a scalable and cost-effective manner. These platforms offer a wide range of tools and services for managing infrastructure, deploying applications, and monitoring performance.

II. Self-hosted deployment: Alternatively, we can deploy the application on a dedicated server or virtual machine using tools like Docker, Kubernetes, or Ansible. This gives us more control over the infrastructure and allows us to customize the deployment environment to our specific needs.

Development and Deployment Environment: In terms of the development environment, we need to set up a local environment where we can develop and test the FastAPI application before deploying it to staging or production. This environment should include the necessary dependencies and tools, such as Python, the FastAPI framework, and a database.

We can use tools like virtual environments or Docker to create a reproducible development environment that closely matches the staging and production environments. This helps to ensure that the application works as expected across all stages of the development lifecycle.

Planned deployment environment: The planned deployment environment should be designed to meet the specific requirements of the project, including performance, scalability, and security. This might involve setting up load balancers, auto-scaling groups, or implementing security measures like SSL/TLS encryption or access controls.

VII.3. Database

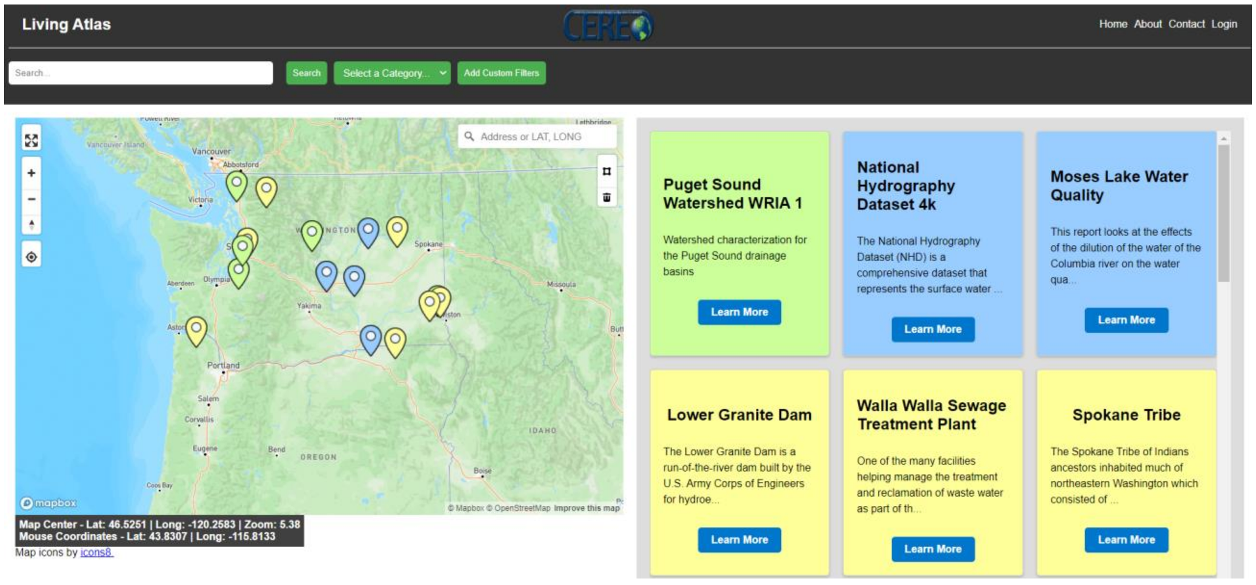
Staging: To stage the SQL database hosted on ElephantSQL for acceptance testing, we can create a separate testing environment that replicates the production environment. This environment should contain a copy of the production data and all the necessary database infrastructure and dependencies required for the application to run. We can use database migration tools like Alembic or Flyway to manage schema changes and seed data for the testing environment. For the production deployment of the database, we plan to use the paid tier of ElephantSQL or a similar database-as-a-service provider. This will ensure that the database is highly available, scalable, and secure.

Deployment: The development environment for the SQL database can be set up locally using tools like SQL clients such as pgAdmin or DBeaver. We can also use ORMs like SQLAlchemy to abstract the database operations and make it easier to write and test database code. We use Git for version control and have set up a remote Git repository to manage the database schema and data scripts.

Development and Deployment Environment: For the deployment environment, we plan to use Docker to containerize the database and its dependencies. This will ensure that the deployment process is reproducible, and that the database can be easily moved between different environments. We can also use CI/CD tools like Jenkins, TravisCI, or CircleCI to automate the deployment and testing process.

VII. Description of the Final Prototype

The Living Atlas App is a repository of environmental information catered to the needs and wants of CEREO. It features a map with points relating to information uploaded by users, and cards representing the details of each of those points. The information stored in the app can range from locations of interest to papers relating to environmental concerns, such as water pollutants.



Map Center - Lat: 46.5251 | Long: -120.2583 | Zoom: 5.38
Mouse Coordinates - Lat: 43.8307 | Long: -115.8133
Map icons by icons8

Puget Sound Watershed WRIA 1
Watershed characterization for the Puget Sound drainage basins

National Hydrography Dataset 4k
The National Hydrography Dataset (NHD) is a comprehensive dataset that represents the surface water ...

Moses Lake Water Quality
This report looks at the effects of the dilution of the water of the Columbia river on the water qua...

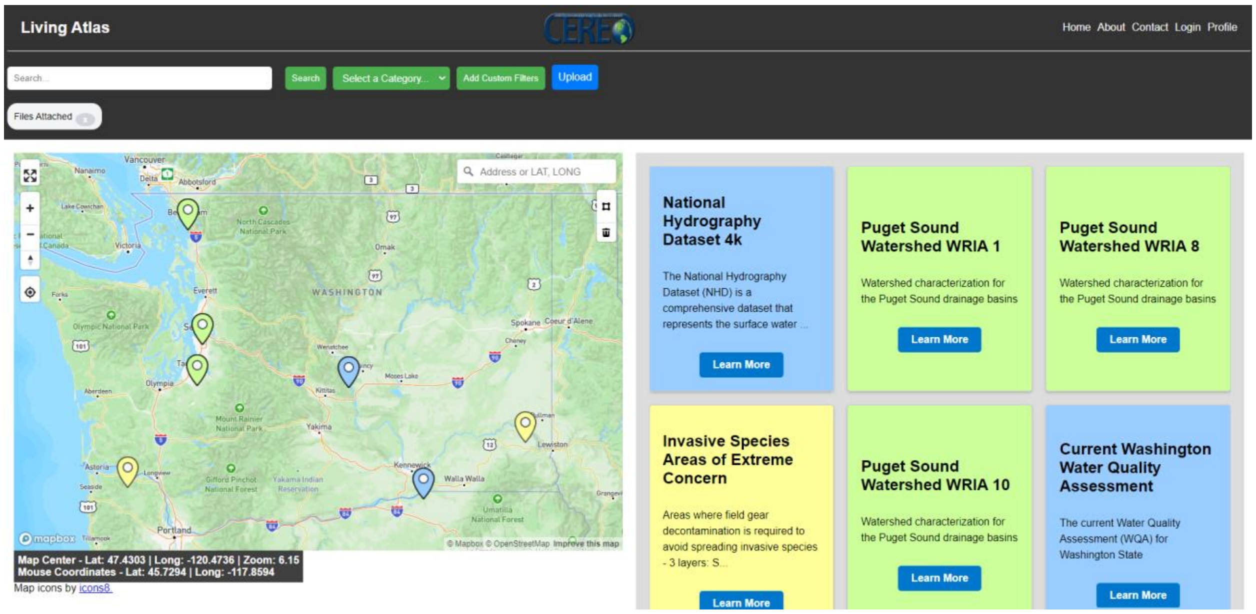
Lower Granite Dam
The Lower Granite Dam is a run-of-the-river dam built by the U.S. Army Corps of Engineers for hydroe...

Walla Walla Sewage Treatment Plant
One of the many facilities helping manage the treatment and reclamation of waste water as part of th...

Spokane Tribe
The Spokane Tribe of Indians ancestors inhabited much of northeastern Washington which consisted of ...

A screenshot from the app featuring the map and cards

When in the app, users can filter the cards by location, which is achieved through moving the map, and zooming in on areas of interest. Each card and point is also color-coded to one of 3 categories: River (blue), Watershed (green), or Place (yellow). Users, when uploading new points and cards, can also add custom tags. Parties searching for certain information can then search using these custom tags. Cards and points can also be clicked on the bring up extra information, including these custom tags. Users can also search for specific cards by using the search bar and typing in what they want to find.



National Hydrography Dataset 4k

The National Hydrography Dataset (NHD) is a comprehensive dataset that represents the surface water ...

Puget Sound Watershed WRIA 1

Watershed characterization for the Puget Sound drainage basins

Puget Sound Watershed WRIA 8

Watershed characterization for the Puget Sound drainage basins

Invasive Species Areas of Extreme Concern

Areas where field gear decontamination is required to avoid spreading invasive species - 3 layers: S...

Puget Sound Watershed WRIA 10

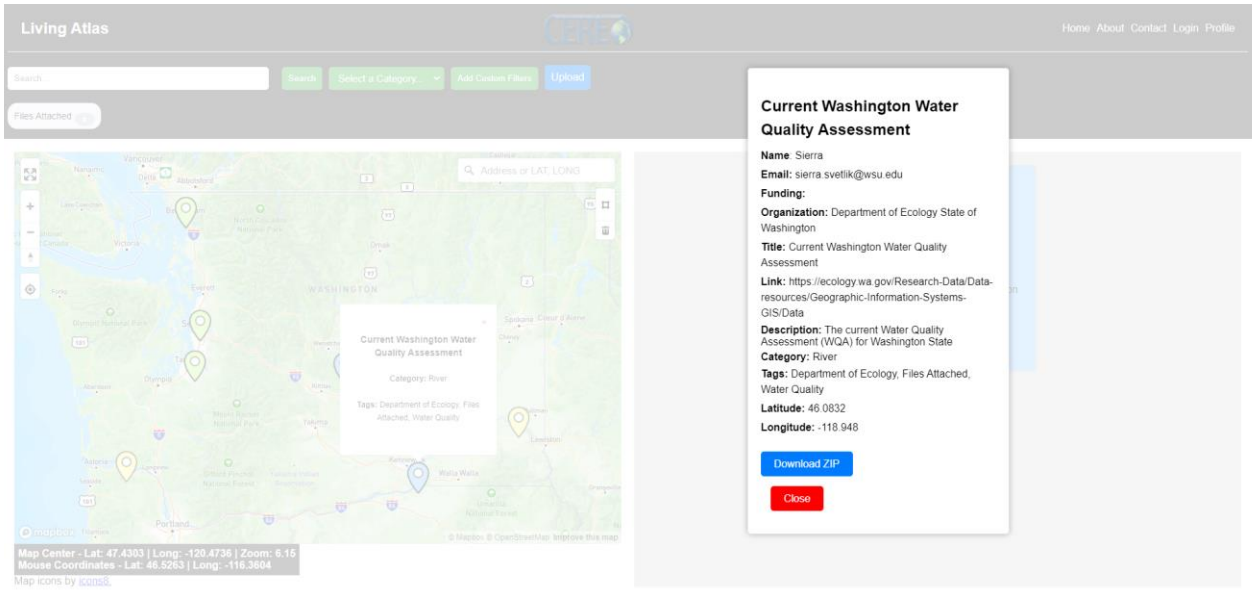
Watershed characterization for the Puget Sound drainage basins

Current Washington Water Quality Assessment

The current Water Quality Assessment (WQA) for Washington State

Map Center - Lat: 47.4303 | Long: -120.4736 | Zoom: 6.15
Mouse Coordinates - Lat: 45.7294 | Long: -117.8594
Map icons by icons8.

The custom filter "Files Attached" has reduced the number of cards down to the ones with associated files



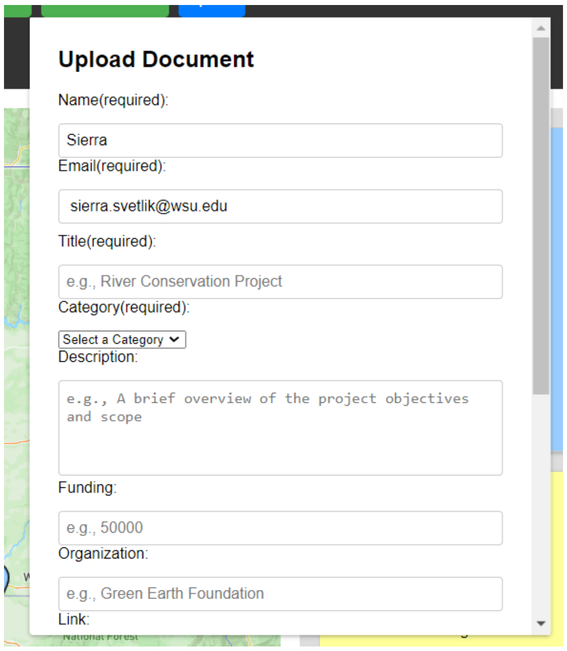
Current Washington Water Quality Assessment

Name: Sierra
Email: sierra.svetlik@wsu.edu
Funding:
Organization: Department of Ecology State of Washington
Title: Current Washington Water Quality Assessment
Link: https://ecology.wa.gov/Research-Data/Data-resources/Geographic-Information-Systems-GIS/Data
Description: The current Water Quality Assessment (WQA) for Washington State
Category: River
Tags: Department of Ecology, Files Attached, Water Quality
Latitude: 46.0832
Longitude: -118.948

Map Center - Lat: 47.4303 | Long: -120.4736 | Zoom: 6.15
Mouse Coordinates - Lat: 46.5263 | Long: -116.3604
Map icons by icons8.

A point and card have been clicked, revealing more information. An option to download an attached file can also be seen

Only logged-in users can upload cards, and, at the moment, only the user who uploaded a card can delete it. When a user wants to upload a card, they click on the upload button, and a menu pops up that contains various fields to be filled in. Not all the fields are necessary, just the title, the latitude and longitude, and the category. Name and email are automatically filled in. Optionally, a description, link to a website, organization, source of funding, and custom tags can be added. Files can also be added.



Upload Document

Name(required): Sierra

Email(required): sierra.svetlik@wsu.edu

Title(required): e.g., River Conservation Project

Category(required): Select a Category

Description: e.g., A brief overview of the project objectives and scope

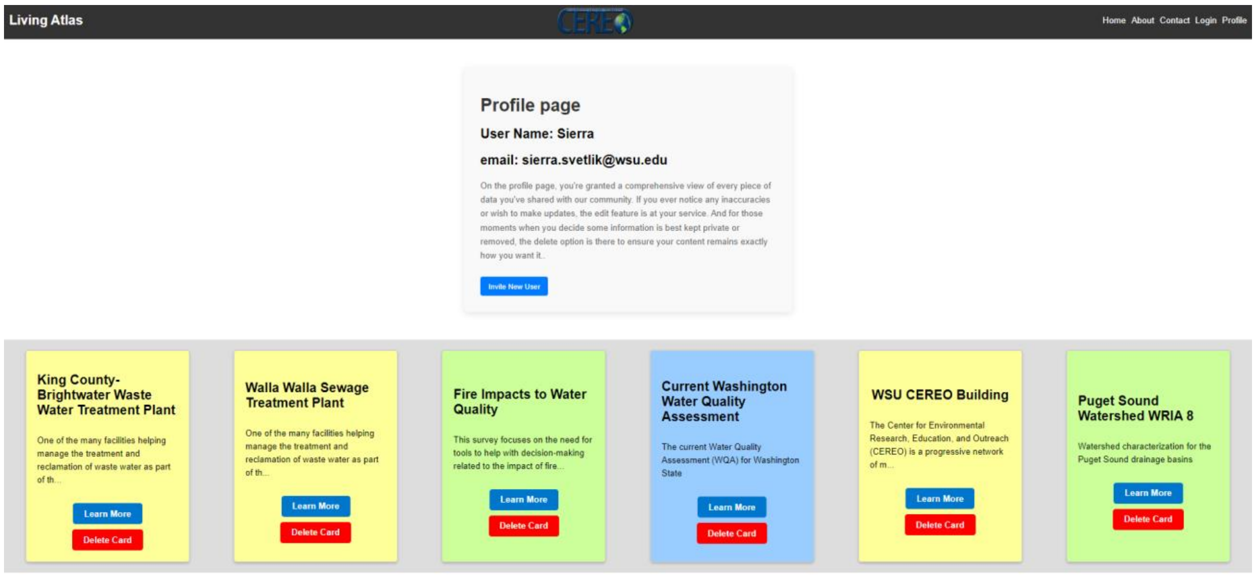
Funding: e.g., 50000

Organization: e.g., Green Earth Foundation

Link:

The upload document dialogue. Required fields are marked

On the user's profile page, they can also see and delete cards that they have uploaded to the app.



Profile page

User Name: Sierra

email: sierra.svetlik@wsu.edu

On the profile page, you're granted a comprehensive view of every piece of data you've shared with our community. If you ever notice any inaccuracies or wish to make updates, the edit feature is at your service. And for those moments when you decide some information is best kept private or removed, the delete option is there to ensure your content remains exactly how you want it..

King County-Brightwater Waste Water Treatment Plant

One of the many facilities helping manage the treatment and reclamation of waste water as part of th...

Walla Walla Sewage Treatment Plant

One of the many facilities helping manage the treatment and reclamation of waste water as part of th...

Fire Impacts to Water Quality

This survey focuses on the need for tools to help with decision-making related to the impact of fire...

Current Washington Water Quality Assessment

The current Water Quality Assessment (WQA) for Washington State

WSU CEREO Building

The Center for Environmental Research, Education, and Outreach (CEREO) is a progressive network of m...

Puget Sound Watershed WRIA 8

Watershed characterization for the Puget Sound drainage basins

A user's profile page, with all their cards. The delete button can be seen on each card

Also on the user's profile page, they can invite new users to the app. All users are trusted to be responsible in using the app and when letting new people create an account.

There is also a contact page for CEREO and an about page for a basic description of the app and its purpose.

VIII. Final Prototype Demonstration

a. Front-End

In the alpha prototype of Living Atlas, we have created a basic layout for the website, with a navigation menu, header, and footer. We have also implemented a card that uses a GET request to dynamically fetch data from the backend and display it in the proper format. Additionally, we have implemented an upload form that uses a POST request to send data to the backend for storage and processing. These features demonstrate the core functionality of Living Atlas and showcase the potential of the application to become a powerful tool for managing and visualizing environmental data.

b. Back-End

The back end for the Living Atlas website involves creating and managing APIs that handle data and logic on the server side of an application. Endpoints are a key concept in this process, they define the various operations that the API can perform and specify the expected input and output formats. I have been using a framework called Fast API which allows me to write these endpoints. Endpoints are created by defining functions that are decorated with specific HTTP methods such as GET, POST, PUT, DELETE, etc. and a URL path. These functions handle incoming requests by extracting data from the request body or query parameters, performing any necessary processing or validation or querying the database that we have, and returning a response in the form of JSON or other formats to the data can appear on the front ed portion we have also been developing. We have also developed a file server as a way to locally store file submitted by the users that are attached to each data point in the card database.

c. Database

The database currently stores, within a user's table, the username, the password in a hashed form, the email, as well as an automatically generated salt and unique user ID. The database also contains a card table that stores a unique data ID, a link to the data, the date that the data was uploaded to the database, a title for the data, a description of the data, and the user ID of the user that uploaded the data. It can also store a latitude and longitude associated with the data, the source of funding, and the organization that the user associates with. There is also a tags table that stores, in each row, a data ID and a tag associated with the data. The tag represents a category that the user thinks is appropriate for the data, and there is one row for each data and tag grouping.

IX. Project and Tools Used

Tool/library/framework	Purpose
React	Frontend framework ideal for web applications with frequently changing UIs

FastAPI	Backend framework that can be developed using python
ElephantSQL	stores all the information for the app on cloud

X. Product Delivery Status

The final product of the project will be delivered in a compressed zip file format, ensuring easy distribution and installation. Alongside this, an active public link will be provided, granting seamless access to the project for all intended users. Additionally, to maintain the integrity and functionality of the software, a comprehensive bug report system will be implemented, allowing the client to report any issues encountered. Moreover, a detailed Payment and Upkeep Report will be prepared, outlining the financial aspects of the project. This report will offer valuable insights into the ongoing costs associated with the project, also including details about passwords and access, helping to ensure transparency and assisting in future budgeting and resource planning.

Project Location information:

1. The latest release of the project is available under Releases in the team's GitHub:

   https://github.com/WSUCapstoneS2023/LivingAtlas1

2. The Link to the latest version of the Front-End:

   https://verdant-beignet-476e34.netlify.app/

3. The Link to the latest version of the Back-End:

   https://livingatlasbackend-36wl.onrender.com/

Set Up Instructions:

Start by cloning the project repository from its designated source. Ensure you have Node.js and npm installed for handling the React frontend. After cloning, navigate to the project directory and run npm install to install all necessary dependencies for React. For the backend, ensure you have Python and FastAPI installed. Navigate to the backend directory and install the required Python dependencies, typically using pip install -r requirements.txt. Then, start the React frontend server using npm start and the FastAPI server by running the appropriate Python script.

XI. Conclusions & Future Work

a. Front-End

In future work, we plan to enhance the functionality of Living Atlas by enabling user login, which will allow users to securely access and manage their data. We also plan to enable file handling, which will allow users to upload and download data files in a variety of formats. In addition, we will enable filtering, which will enable users to sort and search data based on specific criteria. Finally, we plan to integrate a visualization map, which will provide users with an interactive map-based view of the data.

b. Back-End

For the future of the back end, we would want to create more card moderation tools/endpoints. This would include the ability to edit already existing cards and have an endpoint to verify admin users and to have those users be able to delete any card. Another feature the backend would like to implement is the ability to view/download only files. Currently files are attached to cards which require a card to be made but we could add the feature of a file explorer page where all the files are listed, and the user can download any file they desire.

c. Database

For the future of the database, all that will need to be done will be to manage the tables as needed, expanding and adding new tables if it is necessary to store more information. The instance used to run the database will also need to be upgraded should a large number of people start using the app.

XII. Repository information

a. Main Branch

a) Repo URL

https://github.com/WSUCapstoneS2023/LivingAtlas1

b) Last commit information

(Date) 12/10/2023

(hash) 7cca8bc6480e4c612420f1f99f0f05d97e0abe49 **–** f-alvarezpenate

(URL)
https://github.com/WSUCapstoneS2023/LivingAtlas1/commit/7cca8bc6480e4c612420f1f99f0f05d97e0abe49

(Description)

"added db dump file and reorganized db files"

c) Contributor

Joshua Long - joshmainac

Kolb, Mitchell, William - Mitchell-kolb

Svetlik, Sierra Amelia - SierraSv

Wyatt Croucher - WyattCroucher3

Phearak Both Bunna - Phearakbothbunna

Flavio Alvarez Penate - f-alvarezpenate

XIII. Glossary, Reference & Appendices

Axios: Axios is a Javascript library used for making HTTP requests. It provides an easy-to-use interface for making requests to a server and handling the responses.

CLA: CLA is short for CEREO Living Atlas

ElephantSQL: ElephantSQL is a database hosting service that sells databases for customers to use remotely, instead of having all the information stored on local devices.

PostgreSQL: PostgreSQL is a database management service that allows the user to create and modify tables, as well as manage usage of the tables and many other details.

Psycopg2: Psycopg2 is a python package for accessing and modifying PostgreSQL databases.

React: React is a popular Javascript framework used for building user interfaces. It provides a declarative syntax for creating components that manage their own state and render based on that state. React uses a Virtual DOM to efficiently update the UI and provides a way to modularize code through the use of components.

FastAPI: The framework that backend will be using to develop its endpoints to be